NPS-53GP72111A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DATA STRUCTURES FOR QUESTION ANSWERING SYSTEMS

by

Gregory Dean Gibbons

Approved for public release; distribution unlimited.

Data Structures for Question Answering Systems

by

Gregory Dean Gibbons

November 1972

This work was sponsored by the Natural Language Development Project, Advanced Software Technology Division, Navy Electronics Laboratory Center, under Work Request Number WR2-9100.

ABSTRACT:

Data bases for question answering systems are models of reality. Such models can be considered to be sets of assertions. Sophisticated models must cope with consistency problems due to the Frame Problem. Processes that derive answers to questions should be included in the model, and question answering systems should be able to discuss such processes. Uniform processes are too rigid to support intelligent question answering systems. Further understanding of nonuniform processes may derive from development of nondeterministic programming languages. The SIR model, the CONVERSE model, and the DEACON model are compared. Directions for research are proposed.

This task was supported by: Navy Electronics Laboratory
Contract No. WR2-9100

TABLE OF CONTENTS

INTRODUCTION

The computer offers the possibility of automating information and fact retrieval systems. Eventually we will be able to speak directly to computer systems in natural language, explaining what we require of them much as we would outline a task for a human assistant, and engaging in dialogues with them to assure that they (or we) understand whatever is necessary. Current technology supports the implementation of only limited applications of question answering systems; the query languages, subject matter, means of communication, and amount of data are all stringently limited. However, research on question answering systems and problem solving systems has led to some understanding of the linguistic and cognitive processes that must be developed in order to expand the capabilities of fact retrival systems beyond the limits of current technology.

Current question answering systems are able to accept queries stated in more or less natural English, and answer them by referring to a data base. The type of information recovered by such systems could also be obtained by performing various counting and sorting manipulations on data in the form of tables and lists. Causal, qualitative, judgmental, and, generally, deductive information simply cannot be provided by these systems.

Systems designed to deal with specific, limited domains of information can be built with considerable knowledge of the domain incorporated in the design. These special-purpose systems sometimes provide some deductive abilities, and may rival or exceed human performance in deduction and problem solving within their restricted areas. However, such general question answering systems as are available possess very limited capabilities. In addition, natural language processors typically consume large amounts of time and memory and can operate only with relatively small data bases. The problem of coping with large and diversified data bases is receiving some attention, but cannot be considered solved.

With the development of improved natural language technology and related technologies, such as computer speech recognition and production, and computer problem solving, it will eventually become possible to automate most information and fact retrieval systems; but when these developments will occur, and indeed, precisely what developments are necessary, are not completely apparent at this time.

This paper presents an examination of some of the alternatives available for the representation and manipulation of data in question answering systems, together with a discussion of limitations imposed by uniform processes, and compares three specific types of data representation taken from three contemporary question answering systems.

MODELS

A question answering system must answer factual questions. To obtain the necessary facts, the system must have access to a supply of information; such a store of information is a model of reality. In an applied system, factual information about the world would be given to the system to be used in answering questions.

Figure 1 shows some possibilities for structuring a question answering system's model of reality. The model reflects only the relevant aspects of reality - certainly not all of reality. In Figure 1, the first major distinction occurs between direct, mechanical models and formal models. Conceptually, a direct model operates by mechanically reproducing the relevant features of reality. For example, consider the task of proving theorems in elementary geometry. One way of recognizing reasonable hypotheses is to draw diagrams that represent the hypothesis under consideration. Examination of such drawings often reveals the absurdity of an incorrect hypothesis, which can then be dropped; effort that might have been spent attempting to prove the hypothesis correct can thus be saved.

Little is known about direct models, and no question answering system uses such a model. An interesting research project might attempt to develop a direct model for some reasonable subject and to use the model as the basis of a question answering system.

The alternative to a direct model is a formal model. A formal model is one in which the relevant aspects of reality are represented

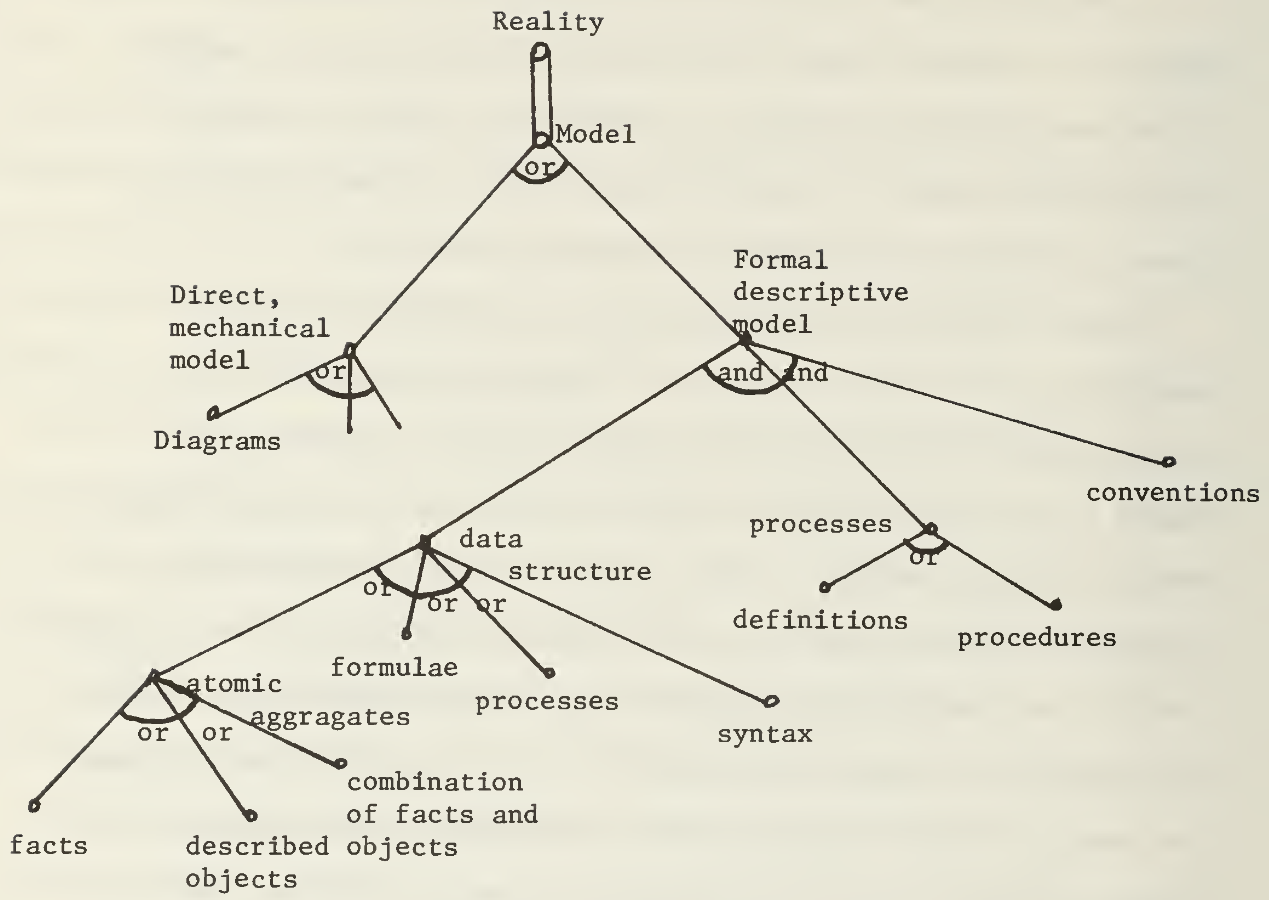

Figure 1. Some alternatives in the construction of models of reality for use by question answering systems.

as assertions in some suitable language. Directed graphs, tables, lists, and various logical calculi are examples of such languages. The prime advantage of formal models is that the related languages are easy to represent in a computer and are relatively easy to manipulate.*

The main disadvantage of formal models is that logically or mechanically necessary properties of reality are not logically or mechanically necessary in formal models, and must be included as additional information. Thus, for example, if you attempt to draw a triangle whose interior angles sum to less than 180 degrees, you will find that you must bend one of the supposedly straight sides, whereas if you simply assert in a suitable language that a particular triangle has less than 180 degrees the error may never be discovered. This issue is related to the frame problem, discussed in the following section.

Since current question answering systems use only formal models, the remainder of this discussion will be restricted to formal models. A formal model consists of a data structure, or language, a set of processes for manipulating the data, and a set of conventions for interpreting the meaning of the data. For example, the data might consist of atomic symbols and lists containing pairs of elements, the

---

* The representation of very large data bases in formal models is not well in hand, but would presumably be even more difficult using a direct model, since direct models are not understood at all.

convention being that the list consist of attribute-value pairs, and the processes being suitable functions for building, changing, and reading such data structures. More abstract conventions might be that the atomic symbols represent atomic quantities such as numbers, names, and fundamental relations, and that objects and sets are represented in the data as description lists.

Data structures may be divided into the following categories: atomic aggregates, formulas, processes, and syntax. (See Figure 1.) Atomic aggregates consist of atomic symbols and structures composed of atomic symbols. For example, numbers, character strings, arrays, and lists can be considered atomic aggregates. Formulas are typically in some logical calculus such as a form of the predicate calculus. Processes can be used as data objects in some languages, such as LISP and PLANNER. For example, Winograd (1971) uses PLANNER to represent his data as processes. Thompson's REL system (Dostert, 1971) stores considerable information in an expandable syntax. This feature of REL will be discussed presently.

There is an important distinction to be made within the category of atomic aggregates. Typically the atomic symbols represent basic quantities, such as numbers and basic relations (e.g., set membership, equality). However, the higher level structures may represent facts, objects, or combination of the two. For example, the fact that John is six feet tall could be represented as (HEIGHT JOHN 6'), and the object John, whose wife is Mary, could be represented using description lists as:

```
2254  (NAME JOHN  WIFE 2642)
2642  (NAME MARY  HUSBAND 2254).
```

Clearly, if the data elements are facts, then the real world objects are represented indirectly via interconnections between facts, and conversely. Ring structures allow both types of connections to be made explicitly in the same data structure, at the cost of considerably increased linkage. We shall presently see examples of all three types of atomic aggregate representations.

Most current question answering systems are based on data structures primarily consisting of atomic aggregates. These structures are less expressive and flexible than formulas, for example, but by virtue of these very qualities, they are easier to manage. We will address two of the factors that limit the development of more sophisticated question answering capability - the frame problem and the lack of nonuniform processes. The frame problem is more critical in highly sophisticated data representations and where changing situations are represented. The liabilities of uniform processes are sometimes subtle, but pervade current systems.

The Frame Problem

The frame problem consists of keeping track of the effects of actions. The word "frame" refers to a frame of reference, so that the frame problem is effectively the problem of appropriately modifying a frame of reference. The frame problem originates in the study of computer problem solving where the machine's task is to construct a sequence of operations that transforms a given initial situation into some desired situation. For example, the reknowned

Missionaries and Cannibals problem consists of an initial situation in which three missionaries, three cannibals and a two-man boat are on the left side of a river. The desired situation is that all be on the right side of the river, and the legal transformations include all the obvious possible actions, except that at no time may any missionaries be left outnumbered by cannibals.

While the frame problem may appear simple, it is extremely refractory. If the frame of reference represents a physical situation, and the modification is due to some physical action having occurred, then it is quite difficult to know just how to modify the frame of reference. For instance, if I have just constructed a color television from a kit, and I plug it in and turn on the switch, how do I know what will happen? On a simpler level, if I am working on the Missionary problem and I transport a missionary, a cannibal, and the boat across the river, how do I know where the remaining missionaries are? While the latter problem is easy, the former is not; further, if the former is not difficult enough, consider attempting to determine how well the contry's missile defenses would work without actually using them in combat.

Approaches to the Frame Problem. There are several methods proposed for handling the frame problem; see Raphael (1971) or Hayes (1971). The most direct is simply to state physical actions as mappings from complete state descriptions to complete state descriptions. Thus, in the Missionary problem, the action of moving the two people across the river would contain also the action of leaving the others where they are. This approach is unworkable if there are many actions or many elements in the complete state description.

Most of the approaches discussed in the literature seem to be directed at solving the problem completely, in the sense that the updating of the frame of reference be done without error and without omission, as much as possible. Hayes (1971), for instance, presents a modification of the predicate calculus intended to allow updating of frames of reference represented as sets of predicate calculus assertions.

Our view is that: 1) a complete solution to the frame problem is too much to expect. Humans do not have anything like a complete solution for the frame problem, since they are routinely surprised or disappointed at the consequences of their actions, and cannot plan far ahead with much detail and reliability. 2) Much expertise about the world is contained in assertions about the consequences of actions, so that it is appropriate to represent the calculation of consequences of actions in the same form as the actions themselves are represented. Consequently, a more pragmatic approach seems appropriate. For example, the machine could alternate between making actions (in which the direct consequences of the actions are recorded in the frame of reference), and updating the frame of reference. In the update phase, the indirect consequences of the action taken could be computed and recorded.

The importance of the frame problem will increase as question answering systems are expanded to provide deductive and problem solving capabilities and to cope with data involving changing facts.

Question Answering Processes

In most question answering systems, the question answering processes are constructed as integral parts of the system, sometimes being fully specified early in the design of the system. Consequently, these processes are not within the system's model. A further practical consequence is that the processes are not clearly and distinctly defined; the major processes make sense only within the total system design, and cannot be understood or discussed effectively out of that context.

If at least some of the question answering processes could be shifted into the model, it might be possible for the system to accept changes and improvements as additional data. Those processes that are not in the model can be changed only by reprogramming. The possibility that a system could participate in generating improvements by a process of verbal reasoning suggests interesting directions for research.

Processes are programs. Programs may be instruction oriented, as most computer languages are. Instruction oriented programs specify sequences of elementary actions, and are typically difficult to treat as data objects; early list processors such as IPL-V had as one of their advertised advantages the fact that programs could be treated as data. Alternatively, programs may specify definitions. In this case, program execution is accomplished by evaluating definitions. LISP is the prime example of such a language. Processes stated definitionally are less distinct from data than are processes stated instructionally.

Both instructional and definitional languages present difficulties to the programmer: in typical languages the programmer must specify minutely the procedure to be followed. By comparison, in giving instructions to a human assistant, a person concentrates on the important or tricky steps, and fills in other details as needed. Usually a human assistant requires far fewer details than a computer.

Higher level programming languages allow the programmer to avoid specifying as much detail as lower level languages require; indeed, this is one of their major advantages. However, the more abstract the language is from the details of the computation, the more decisions the language processor is required to make. Language processors approach each program in the same way, regardless of whether the program is large or small, sophisticated or elementary, or how long it will run. The language processor applies just the same procedure in each case. Such a processor is referred to as <u>uniform</u>. Current language processors and most theorem proving programs are uniform processes.

The Liabilities of Uniform Processes

For an example of the drawbacks of uniform processes, consider the factorial function and the Fibonacci sequence. Suitable recursive definitions of the factorial function and a function FIB(N) which computes the nth Fibonacci number are given in Figure 2. If a typical LISP interpreter were given these definitions, it would require $N + 1$ applications of the definition to compute N factorial. However, since each application of the definition of FIB in which N is greater than 1 requires two additional applications of the definition, the calculation of FIB(N) generates a binary tree whose minimum depth is $1 + N/2$ and whose maximum depth is N. Each node represents an application of the definition; since there are more than $2^{(1 + N/2)}$ nodes in this tree, the time required to calculate FIB(N) is at least proportional to $2^{(1 + N/2)}$. For example, if each application of the definition took 25 micro seconds, the time required to calculate FIB(64) would be greater than $25*10^{-6}$ sec. * $2^{(1 + 64/2)}$. Now $2^{(1 + 64/2)} = 2^{33}$, which is approximately 8 billion, so the time estimate is roughly $25*10^{-6}*8*10^{9}$ sec., or 200,000 seconds. While no one would attempt to calculate N factorial faster than the LISP interpreter, any college calculus student should be able to calculate the 64th Fibonacci number in well under 200,000 seconds. Of course, a simple modification in the program or the interpreter would render the FIB function as fast as the factorial function, but that is just the point: the rigidity of a uniform

```
FACT(N) = if N = 0  then 1  else N*FACT(N - 1).


FIB(N)  = if N = 0  then 0;
          if N = 1  then 1;
          else FIB(N - 1) + FIB(N - 2).
```

Figure 2. Recursive definitions of the factorial function and a function to compute the nth Fibonacci number.

process prevents it from taking advantage of local opportunities, or, put another way, leaves it vulnerable to unforeseen troubles like the combinatorial explosion just presented.

Currently almost no work is aimed at providing language processors with the ability to analyze the programs they are given so that they might, for example, calculate Fibonacci numbers from the recursive definition without falling into the trap just presented.*

There are two primary reasons for the lack of nonuniform processors. First, uniform processes are easier to understand and build. Being understandable, they seem trustworthy, if rigid. Also, where they are sufficient, they are far cheaper in computer power and run much faster. Second, current languages actually leave little decision making to the processor, so that uniform processes are feasible. (This may constitute one of the main differences between natural and artificial languages.) For instance, the Fibonacci example above, while valid, may seem a little strained. In fact, the world of computing does not abound with examples in which a uniform processor runs wild on a simple program.

A flexible, competent information system that communicates with its users in English will necessarily be a nonuniform processor. The rigidity of most current natural language systems results from the fact that they are uniform processes in which many fundamental

---

* Code optimizing compilers do, in a sense, provide some of this ability, but they are themselves uniform processes, and treat simple and complex programs identically.

decisions are made a priori, in the design stage. If the question answering processes could be moved into the system's model, and made the object of conversation with the user, the rigidity could be reduced.

Nondeterministic Programming Languages

It appears that the development of nonuniform processors will derive from work on problem solving. A problem solver is a non-uniform process because, in order to solve nontrivial problems, it must select its actions carefully on the basis of an analysis of its current situation and goals. Of course trivial problems can be solved by simply enumerating possible solutions; but in nontrivial problems far too many possibilities exist. Problems are actually stated as programs in simple nondeterministic programming languages by Fikes (1970) and Gibbons (1972). These programs are in turn executed by nonuniform processes.

Nondeterministic languages offer increased expressiveness over conventional languages, because the programmer can leave substantial decisions to the processor. The processors, in turn, must either make rational decisions for themselves, rely on advice from the programmer or user, or else fall back on an enumeration of possibilities.

The predicate calculus can be used as a nondeterministic programming language. Green (1968) explored the possibility of using the predicate calculus to represent problems, and applying theorem proving techniques to find solutions. This approach is limited by representational difficulties due to the frame problem, and by the noted liabilities of uniform proof procedures.

A very interesting direction in language development, including elements of the predicate calculus, LISP, and the use of advice from the programmer, is represented by PLANNER. PLANNER (Hewitt, 1969) is a nondeterministic programming language that has actually been used to implement major systems such as Winograd's system (see Winograd, 1971, for a description of this system and also for a good description of PLANNER). PLANNER results form the addition of nondeterministic elements to LISP.*

A PLANNER program is written as a set of function definitions, as in LISP. A PLANNER function is normally considered to be a truth function, so that the evaluation of a function results in TRUE or FALSE, with side effects. For example, to solve the problem "find x and y in $(1, 10)$ such that $x + y = 15$," one would write a PLANNER function that represents the predicate calculus expression $(Ex)(Ey)((x + y = 15)$ and $(x \in (1, 10))$ and $(y \in (1, 10)))$. Evaluation of the function would cause the PLANNER system to prove that the expression is true.

The PLANNER interpreter uses advice given in the definitions by the user, but is still a uniform process. The basic mechanism in

* Actually, it is Micro-PLANNER, a preliminary version of PLANNER, that is implemented in LISP. A full PLANNER system is not yet available.

the PLANNER interpreter is a generate and test algorithm. Fundamentally, then, without the addition of further mechanisms to control the search, the interpreter would suffer the practical deficiencies of the generate and test algorithm. The major method of controlling the search is the use of advice supplied by the programmer. This advice is attached to the definitions of functions, and so is scattered throughout the PLANNER program. The user familiar with PLANNER's generate and test procedure can obtain complete control of the system's search by adding appropriate advice. In addition, this advice can be manipulated by the running program, which suggests interesting possibilities for experimentation with learning.

PLANNER, or some other suitable nondeterministic language, would be a good choice of language in which to write question answering processes to be stored in a system's model. The processes in the model could be quite nonuniform, with the only uniform process being the programming language processor. (There is always a uniform process somewhere, just as behind any search process, no matter how sophisticated, there is always a generate and test process.)

Conventions and Types of Data

The question answering capability of a system derives from the capacity of its data structure and the processes that manipulate the data to obtain answers. Mediating between these is a system of conventions in terms of which the data are encoded and the processes written.

Conventions. Each process must know how to interpret its inputs; thus in finding the answer to a question, a program must know the significance of the data it finds. This is accomplished by specifying the program partly in terms of conventions about the data. For example, LISP's EVALQUOTE expects arguments FN and X, where FN is the name of an elementary function or a lambda expression that defines a function, and X is a list of expressions whose values are to be the arguments of FN.

The documentation of question answering systems always contains explanations of the system's conventions for representing information in the data base. For example, one finds discussions that explain, say, that "relation R holds between A and B" (i.e., R(A,B)) is represented as: $A \xrightarrow{R} B$. This is essentially a translation rule: it explains how to translate an assertion in English (or some query language) into an element or portion of a data base.

Two observations are relevant here. First, in isolation, both the source and target representations are equally meaningless. They become meaningful only in a context that contains conventions and processes for interpreting them. Second, conceptually we might as well consider the target representation to be an assertion, so that the formal distinction between a datum and a (definitionally stated) process may be reduced or eliminated.

Types of Data. The data base organization of question answering systems always contain elements that are identified as atomic objects and connections between these objects. This suggests graphs; in fact, labeled, directed graphs are always a suitable, though sometimes unwieldy, basis for representing a system's data structure. Thus, graphs provide a vehicle for comparison of diverse data structures.

On a less atomic level of discussion, the data used by question answering systems can usually be found in some combination of the following categories: atomic aggregates, formulas, processes, and query language syntax.

Atomic Aggregates. Atomic aggregates are data structures consisting of atoms and relations between atoms, where the atoms and relations are constant. These aggregates are usually identified with either facts or object-descriptions. Some network representations appear to combine properties of facts and object-descriptions.

Facts are simple elementary assertions, such as (HUMAN SOCRATES) of (POPULATION PODUNK 120). The defining characteristic of facts as opposed to described objects is that facts are stored explicitly, whereas object descriptions explicitly store objects and their descriptions, leaving various facts implicit. Thus, if we wish to represent the assertions that Socrates is a human philosopher, the fact representation might be:

```
(HUMAN SOCRATES)
(OCCUPATION SOCRATES PHILOSOPHER)
```

Alternatively, we could let a description list stand for Socrates:

```
((NAME SOCRATES)(OCCUPATION PHILOSOPHER)(HUMAN))
```

In addition to specifying what is stored, it is also necessary to specify how it is stored. Facts can be naturally stored as sets or files of n-tuples. For example, a data base containing information on people might include their occupations; this information could be stored in a file, thus:

OCCUPATION	
JONES	PROGRAMMER
SMITH	WRITER
.	.
.	.
SOCRATES	PHILOSOPHER
.	.
.	.

Described objects could be stored individually, with each being part of a lexical entry. Or, for example, lists (representing sets) could be stored in the data base to facilitate searching sets of objects.

Network structures provide the capacity to establish essentially arbitrary connectivity properties in the data base. For example, in the ring structures of DEACON (Craig, et. al., 1966) and REL, lists of objects, N-tuples, and sets of N-tuples all exist in an interconnected structure.

The distinction between the types of atomic aggregates (facts, described objects, and networks) is practical, rather than fundamental. Each data base of atomic aggregates can be unambiguously represented by a set of atomic assertions. The varieties of structure have the practical purpose of establishing direct connections between related assertions, and thereby eliminating the need for searching the whole data base for related assertions.

Formulae. In this context a formula is an expression involving a variable and possibly logical connectives. "All men are mortal" is an example of an assertion that would be represented as a formula. Getting information from formulae normally requires the use of some deductive or at least variable binding process. In a static data base, some formulae could be converted to atomic aggregates by enumerating the instantiations, e.g., tagging every item representing a man with mortal. However, not all the information in a set of formulae can be obtained in this way, and the volume of new data entries would swamp the system. Thus, formulae can be much more concise and expressive than atomic aggregates, but they require the availability of far more elaborate deductive processes.

The deductive processes available today are generally uniform, rendering them impractical as general question answering or problem solving methods. Also the frame problem limits current attempts to represent realistic problem situations with formulas. Cordell Green's thesis (Green, 1969) thoroughly explores these issues, and essentially carries the application of formal (predicate calculus) representation and uniform proof procedures to problem solving and question answering to their limits.

Processes as data. Representing processes in the data base offers the possibility of bridging the gap between problem solving or question answering processes and information in the data base. Including formulae and suitable deductive processes as data is a step in this direction, but the inefficiency of uniform processes poses a serious difficulty. With processes represented as data, it becomes possible to discuss, construct, and revise the represented processes. The uniform process (which is not in the data base) then can function as an interpreter for the processes in the data base, rather than as a problem solving process itself. This is potentially a major source of power for question answering and problem solving systems. With processes available as objects of discussion, it should be possible for the system to engage in verbal reasoning and advice-taking tasks.

Winograd's work is a step toward exploiting these possibilities. His data base is composed of facts and PLANNER theorems. A PLANNER theorem is an assertion, but it also contains advice to the interpreter (uniform process) for evaluating (proving) the assertion. Winograd uses the facilities of PLANNER to obtain reasonably efficient searches in the solution of problems. His system can also discuss its actions, within limits. The addition of advice taking would be a major improvement to the system and would constitute a most interesting development in natural language processing.

Query language syntax. The REL system uses the syntax of the query language to store generic and definitional information. This method has some very interesting properties.

REL uses an exhaustive bottom up parser. One can add rules to the grammer via the query language. Thus, the generic information and definitions are stored essentially as paraphrase. For example, if one defined the factorial function:

```
def:  N! = if N > 0 then N*((N - 1)!) else 1.
```

then a rule would be entered that would replace "i!" by "i*((i - 1)!)" if i > 0 and "1" otherwise. Thus, in response to "3!" the parser would generate:

```
3!
3*((3 - 1)!)
3*( 2! )
3*(2*((2 - 1)!))
3*(2 * (1))
3 * 2
6
```

Generic information could be stored in a similar way, e.g.: Df: city = smoggy place. However, little actual inference can be accomplished because each rule that can be applied is applied. Thus, if we have defined the following:

```
man = adult male human being
being = creature or entity
creature = animate object
adult = fully developed, mature
```

then the sentence "John is a man" would parse into "John is a fully developed, mature male human animate object or entity."

REL achieves some interesting, even spectacular, results using the grammar to allow extensions of the query language; clearly, though, even with the REL's substantial control of the grammar, it is best to store facts in the data base and reserve manipulation of the grammar for the extension of the query language.

COMPARISON OF THREE DATA REPRESENTATIONS

In this section three data structures present in contemporary question answering systems are examined, representing the following three alternatives: 1) data elements represent objects; 2) data elements represent facts; and 3) a combination of the two. Each of the three will be used to represent a sample data base which, though small, contains complexities representative of those found in realistic data bases. The sample data is the information in the following assertions:

1) John is a person.
2) The height of John is 6'.
3) John's siblings are A and B.
4) A person has two hands.
5) A finger is part of a person.

The information in these assertions is somewhat ambiguous; for example, in Number 3 it is not clear whether John may or may not have other siblings. In addition, to cope adequately with these assertions, it is necessary to have additional information such as the fact that the sibling relation is symmetric.

A data management system may either represent the data in a way that leaves the ambiguities intact, or use some method to resolve the ambiguities. Typically, the ambiguities are resolved. This can be done by: 1) asking the user, 2) using a built-in criterion or deduction to decide, or 3) failing to recognize the ambiguity at all, and simply storing one unambiguous interpretation.

The SIR Model

In the SIR system, (Raphael, 1964), words name objects and relations. Objects and relations are associated by relations. These associations are represented by description lists in which each attribute names a relation and the value(s) are other objects or structures that enter into the named relation with the object.

Three types of linkage between objects are allowed:

Type 1 - The attribute has a unique value
Type 2 - The attribute can have multiple values
Type 3 - The attribute can have multiple described values

Notice that Types 1 and 2 are special cases of Type 3, except that with Type 1 there is an implicit assertion that the attribute cannot have more than one value.

In our sample data, heights would be a Type 1 relation, is a and sibling would be Type 2, and subpart (from "has") would be Type 3. Thus, the sample data would be represented:

```
(JOHN (IS A PERSON)  (HEIGHT 6') (SIBLINGS (A B)))
(PERSON (SUBPART ((PLIST NAME HAND NUMBER 2) (PLIST NAME FINGER)) ))
```

Reverse Links. The linkages represented so far are all one-way. To follow inverse relations ("what persons are there?") would require a search of the whole data base, unless reverse links are included. Thus, the data representation becomes:

```
(JOHN (IS A (PERSON)) (HEIGHT 6') (SIBLINGS (A B)))

(PERSON (SUBPART ((PLIST NAME HAND NUMBER 2)
                  (PLIST NAME FINGER)))
        (IS A^-1 (JOHN)) )

(A (SIBLING^-1 (JOHN)) )

(B (SIBLING^-1 (JOHN)) )
```

This includes IS A^{-1} and SIBLING^{-1}, but not HEIGHT^{-1}. Clearly there is a trade-off between storage cost and search cost, so that the selection of reverse links to include should be made carefully. Perhaps it could be done automatically on the basis of frequency of use.

Graph Representation. Now that a specific representation of the sample data has been determined, we can translate it into a directed graph, shown in Figure 3.

The SIR fact retrieval and deduction methods. SIR was intended to make reasonable conversation. To do this, it needed the ability to draw reasonable conclusions from its data, as well as to answer specific questions. Raphael intended to select an internal representation that "---had connectivity and accessability properties---" suitable for such processes. Further, he argues that "...the search programs (must be) designed to "know" the properties of the relations being searched, e.g., transitivity. Therefore a special set of programs had to be written for each relation."

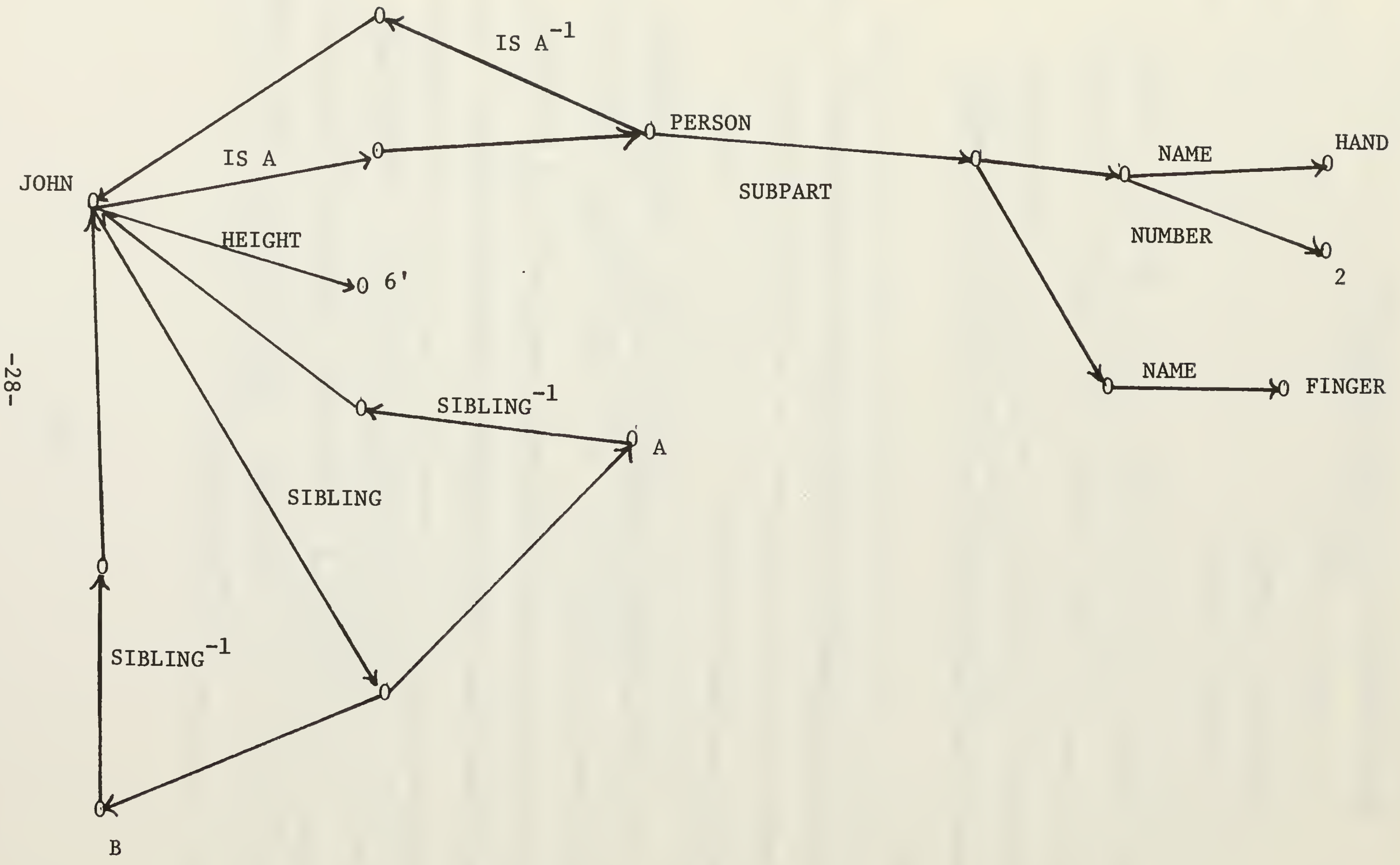

Figure 3. Graph of sample data represented by SIR.

Raphael's argument just quoted is not conclusive. Certainly the easiest way to implement the required deductive processes is by means of special purpose programs that are designed for particular relations. However, it is also possible to construct descriptions of relations, and apply a general deductive process that uses such descriptions. If such descriptions could be expressed in the query language, they could be included in the data base. The general deductive method would function as an interpreter, and the descriptions of relations would function as particular deductive methods. Such a capability should be within reach of current technology.

<u>Relations</u>. Because each relation requires a separate set of programs, SIR was given a limited set of relations. They are:

Set membership	('John is a person')
Part-whole	('A finger is part of a person')
Number	('A person has two hands')
Set inclusion	('A man is a person')
Left-right	('The lamp is to the left of the table')
Ownership	('John owns a car')

The inverses of these relations are also included.

SIR can do several interesting things, such as handle exceptions to generic information in a reasonable way. The search program always looks for information about an object on the object's description list. If the information is not there, the program looks at the descriptions of sets to which the object belongs. See Figure 4 for an example.

SIR would require additional programs for the height relations before it could answer the question "What is John's height?"

Figure 4

```
(***.  THERE ARE 5 FINGERS ON EVERY HAND)
(I UNDERSTAND)

(***.  THERE ARE TWO HANDS ON A PERSON)
(I UNDERSTAND)

(***.  A BOY IS A PERSON)
(I UNDERSTAND)

(***.  TOM IS A BOY)
(I UNDERSTAND)

(***.  DICK IS A BOY)

(***.  HARRY IS A BOY)
(I UNDERSTAND)

(***.  TOM HAS NINE FINGERS)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS
AS PARTS))
(I UNDERSTAND)

(***.  DICK HAS ONE HAND)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS
AS PARTS))
(I UNDERSTAND)

(***.  HOW MANY FINGERS DOES TOM HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS
AS PARTS))
(THE ANSWER IS 9)

(***.  HOW MANY FINGERS DOES DICK HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS
AS PARTS))
(THE ANSWER IS 5)

(***.  HOW MANY FINGERS DOES HARRY HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS
AS PARTS))
(THE ANSWER IS 10)

(***.  HOW MANY FINGERS DOES JOE HAVE Q)
(THE ABOVE SENTENCE IS AMBIGUOUS ** BUT I ASSUME (HAS) MEANS (HAS
AS PARTS))
(I DON'T KNOW WHETHER FINGER IS PART OF JOE)
```

Figure 4. Example of the exception principle in SIR.
(From Raphael, in Minsky, 1968)

The CONVERSE Model

The CONVERSE data base is composed of objects and relations. The data structure is patterned after the logical notion of a finite model. The set of objects is the universe of discourse, say U. If R^n is an n-place relation, then in a finite model $R^n \subset U^n$. CONVERSE would store R^n as the set of n-tuples $\bar{X}$ of objects such that $R^n(\bar{X})$ is true.

A data base containing our sample data would thus appear as:

height = { < JOHN, 6' > ... }

sibling = { < JOHN, A >< JOHN, B > ... }

person = { < JOHN >, ... }

(Here we have taken person as a one-place predicate. We could also use set membership as the predicate referred to in "John is a person", and store ε = {< John, person > ...} . There is no apparent reason to do this, however.)

When we come to the SUBPART relation, in order to store "a person has two hands" and "a finger is part of a person," we would have to use at least three place relation:

SUBPART = {< person, hands, 2 >< person, finger,?> ...}

Here "?" is used as a place holder, to signify that the number of fingers per person is not known. In fact, this is still not quite legitimate in the absence of special search functions, because what is really meant is that anything that is a person must have two hands.

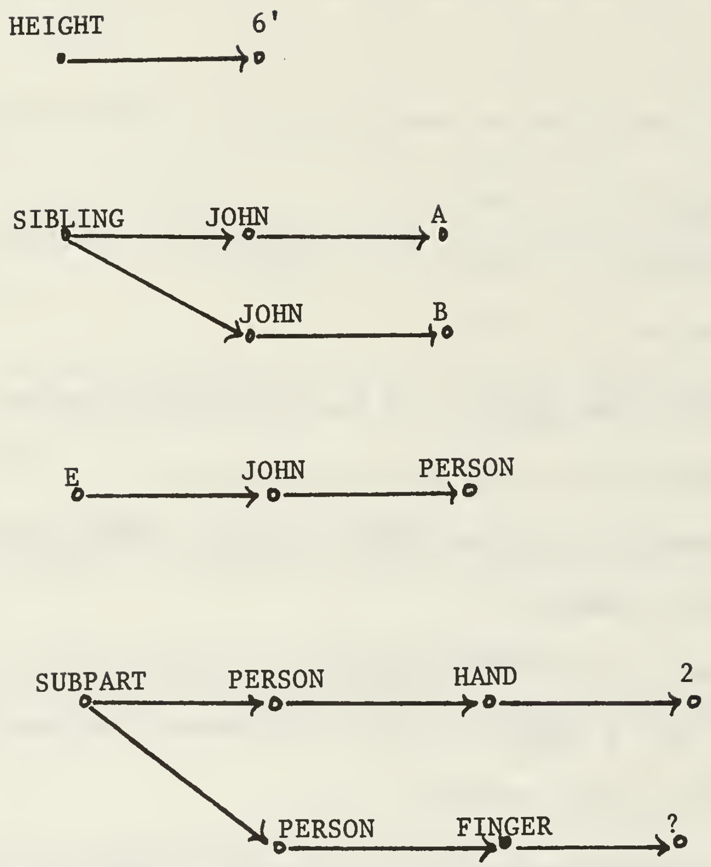

Figure 5. A representation of the sample data in terms of CONVERSE relations.

This assertion could be rendered in the predicate calculus as $\forall x$ (person (X) $\supset$ Subpart (X hands 2)). The interpretation of this formula in a finite model would be that $< x$, hands, $2 >$ would append as an element of the relation SUBPART for each x such that $<x>$ appears in person.*

A reasonable question answering system using this data base organization would use special search functions to avoid the above difficulties. This being the case, the data structure is no longer strictly a finite model.

It is not clear what advantages are to be gained from the finite model structure. The logician's purpose in excluding relations from the universe U is to assure that the finite model is unambiguous, and the truth of an assertion can be calculated directly, without the need for deduction. Properties of predicates such as symmetry of two place predicates are therefore properties of models, rather than elements of models. The question answering system builder's purpose is different: he wants to be able to store and recover facts easily and make at least simple deductions easily.

Since the objectives of the logician are so different from those of the system builder, it would be surprising if a strict finite model were particularly appropriate for question answering. On the contrary, there are some significant disadvantages. For instance, in a finite model, relations are not objects. Thus, the data

* Note that in the finite model, PERSON = $\{< \text{person}_1>, < \text{person}_2>, \ldots\}$

cannot assert directly that a given relation has a certain property - e.g., "sibling is symmetric" or "a person has two hands."

Another disadvantage with CONVERSE relations is that, once specified, they are fixed forever. In ordinary conversation, we feel free to use as many or few modifiers and modalities in our assertions as suits our purpose. If CONVERSE specified a place in each relation for each possible argument, the n-tuples would be long and often full of place holders. Leaving such places out of the relation prevents CONVERSE from using the modality at all. Thus if SUBPART were a two place relation, then the fact that a person has two hands could not be represented. The SUBPART relation would appear as:

SUBPART = {<person, hand><person, finger> ...}

Other systems use data structures that allow the flexibilities mentioned here. Compare, especially, Winograd (1971) or Quillian (1966).

One of the principal advantages claimed for this data structure in the CONVERSE literature is that rapid lookup can be achieved by means of binary search. However, this advantage results from storing long, ordered arrays of similar elements, not from the fact that the elements are n-tuples. This advantage could be secured as well if the elements stored were small networks rather than n-tuples. This would allow for direct storage of modalities:

SUBPART = {<person, finger>,<person, hand, (mode (number 2))> ...}

Such a scheme appears to capture the claimed advantages of the finite model, while serving the question answering goals more directly than a pure finite model can.

Rings

The ring structures considered here were used in the second DEACON system (Craig et. al., 1966), and in the first REL system. The ring structure was developed from the original list-structured data base used in the first DEACON system. The motivations for developing the ring structures were primarily these: (1) over-specialization in the list-structured data base was (wrongly, I believe) blamed on inherent limitations of list structures such as those available in IPL/V and LISP. Ring structures were intended to overcome these supposed limitations. (2) Ring structures provide much more connectedness in the data; this allows the retrieval functions to follow more direct paths within the data.

The rings are implemented as two-dimensional circular lists, i.e., each list word contains a symbol and two links, and the lists are circular in both dimensions. Because of the double linking, the rings cannot be conveniently represented in LISP notation. The simplest direct notation is a graph. The sample data represented in rings is shown in Figure 6.

The solid rings represent objects, sets, and properties (JOHN, PERSON, and HEIGHT, respectively). The dotted rings represent facts, such as (JOHN SIBLING A), (JOHN PERSON). Each dotted arrow carries

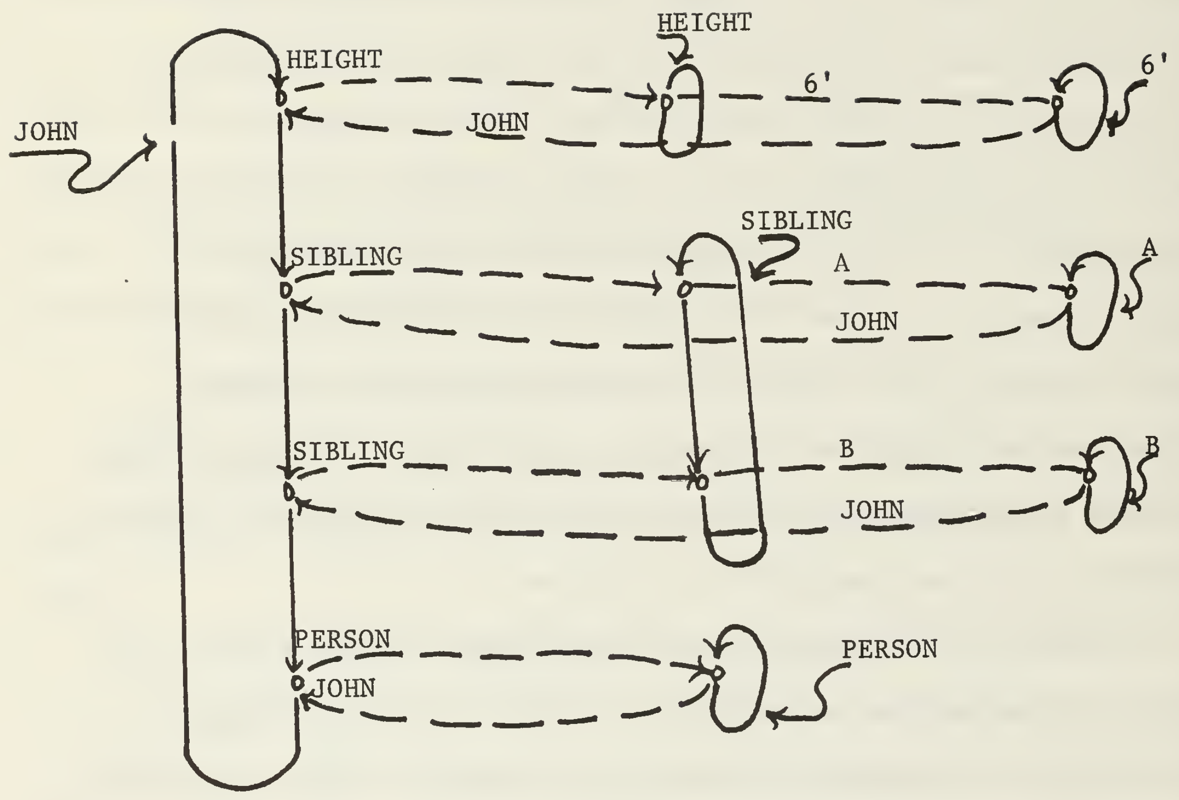

Figure 6. The sample data represented in rings.

the name of the ring to which it points - the arrow itself points to a particular point within the ring.

Note that there are only two types of dotted rings - those of length two and those of length three. The conventional interpretations are (<object><attribute><value>) and (<object><superset>). Consequently, the data base represents the facts in the sample data quite ambigously. Thus, "John is a person" and "person is a John" result in the same data. Such nonsense had to be eliminated at the syntatic level in DEACON and REL.

The basic ring structure represents only elementary facts, and in such a way as to require syntactic help to provide unambiguous interpretation. Consequently, very little a-priori information is contained in the data structure itself. Thus, the relations (such as HEIGHT) are themselves data elements, rather than being parts of the data structure. Another consequence is that no inference capability is included because such capability would require pre-established conventions for interpreting the data. Thus, the transitivity of LOCATION, if it is to be available, must be supplied outside of the data base, and generic information such as "a person has two hands" does not fit into the data base at all.

<u>Generic Information</u>. Within the basic ring structure there is no explicit allowance for generic information. Thus, while one could add a fact (PERSON SUBPART FINGER) to represent "a person has a finger", the data would not show the distinction between "each object that is

a person has a finger" and "the object <u>person</u> has finger". In SIR this type of distinction was handled by a search routine associated with the SUBPART relation; no such individual search functions were used in DEACON or REL.

<u>Time-dependent Information</u>. The retrieval and linguistic methods of DEACON and REL allowed them to cope with assertions about facts that changed with time. To deal with such facts, the ring structure was extended to allow the (<object><attribute><value>) ring to contain a time modifier. The result was that the assertions "the location of A is X from t_o to t_1" and "the location of A is Y after t_2" would be represented as in Figure 7.

These systems contained built-in functions to search such structures. The tense information in a sentence was encoded in terms of markers such as BEGIN and END. Some modest inference ability was included to allow the system to answer "yes" on the basis of the above data if the current time t is greater than t_2, and the question is "Is A at Y?" Neither system provided any approach to the frame problem. The frame problem was never troublesome because the systems made essentially no deductions and the data on which they worked contained no temporal inconsistencies.

<u>Modalities</u>. The assertion "a person has two hands" presents the same difficulties in rings that it did in the CONVERSE data structure. The fact asserted could be taken as a three place relation, i.e.,

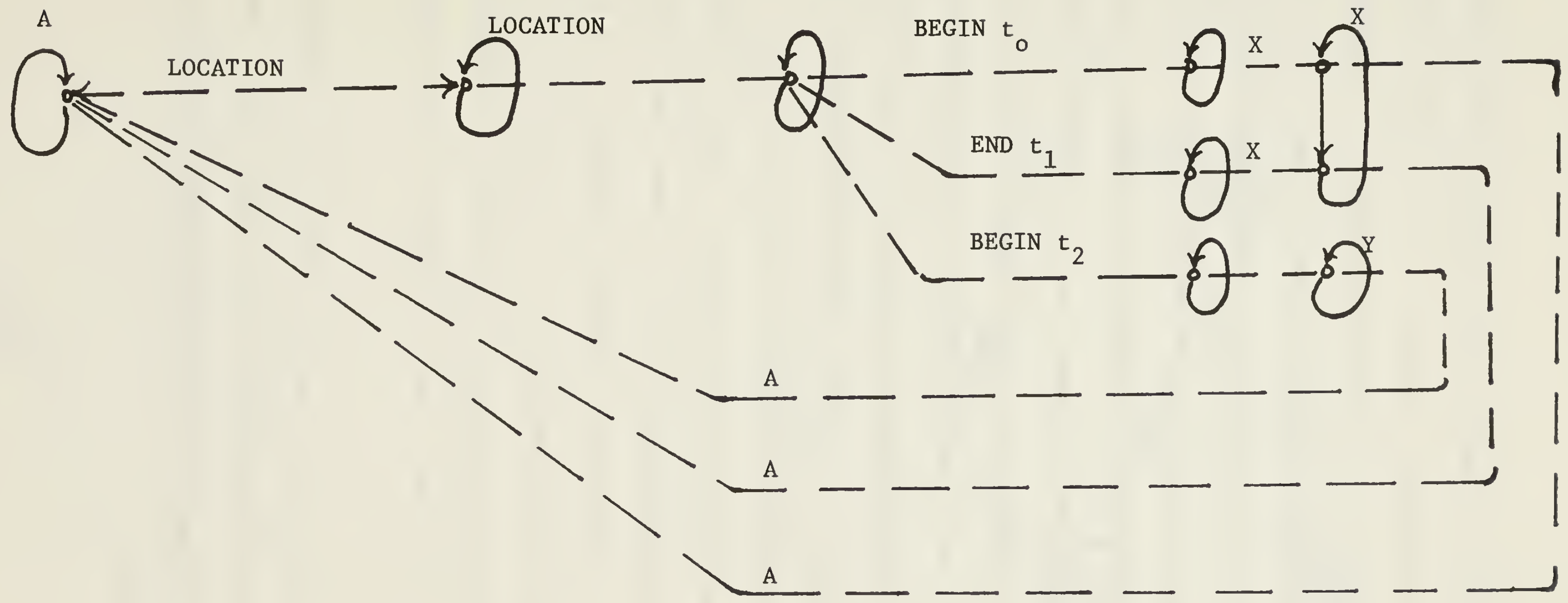

Figure 7. Example of time-dependant information in rings.

(SUBPART PERSON HAND 2); however, such convention would restrict the last place in the relation to a number. A more flexible approach is needed. An extension of ring structures similar to that used to represent time-dependent information could be used to allow modalities, and still avoid the difficulties that arose in the CONVERSE representation. Figures 8 and 9 show two possibilities.

In Figures 8 and 9 the two assertions "a finger is part of a person" and "a person has two hands" are represented. In Figure 8, the two facts are combined in a single complex value for the attribute SUBPART using the same structure that was used to represent time-dependent facts. As this example indicates, the representation of time-dependent facts can be used to contain other types of modal information by extending the (<object><attribute><value>) type ring to be (<object><attribute>((<mode><value>)...(<mode><value>))). Clearly, also, the need for the ability to make simple deductions increases with the elaboration of model information. In Figure 9, simple and complex values are mixed in a different but also plausible way, which illustrates the need for conventions for interpreting a data structure. This need is more stringent for more complicated structures.

The ring structure could support other types of elaborations, such as the part-whole assertions in our sample data. However, such elaborations would require corresponding special storage conventions and search functions.

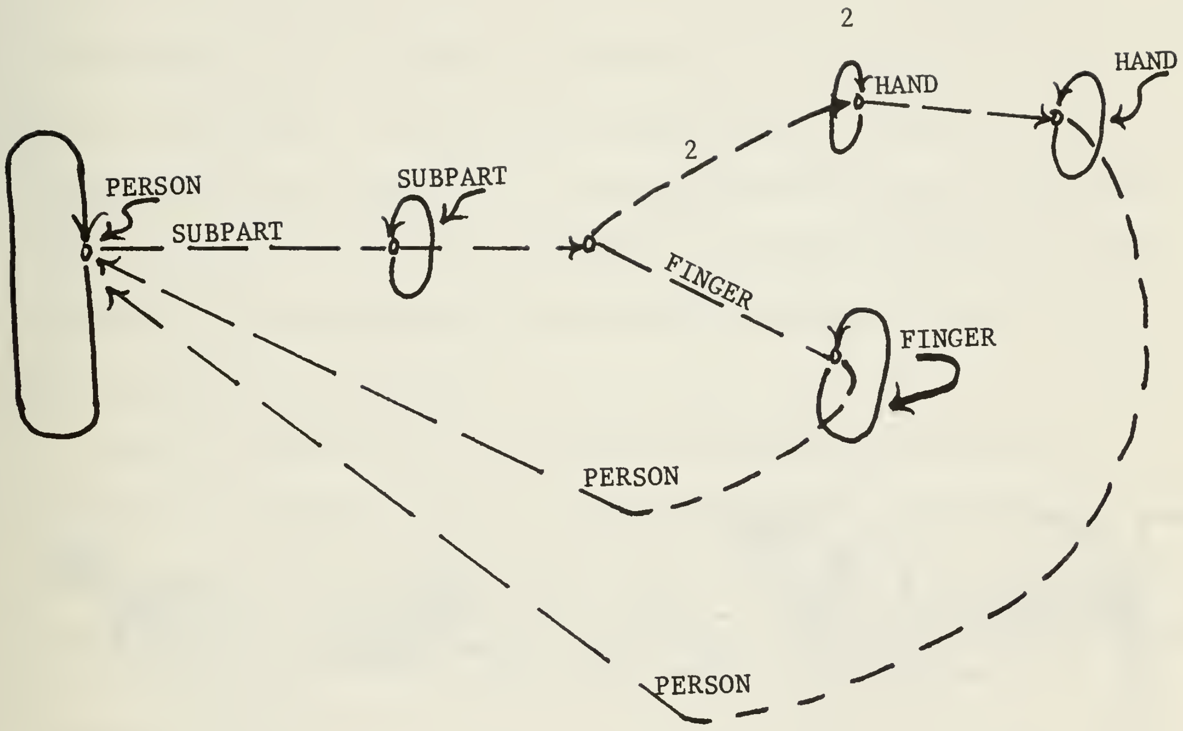

Figure 8. Possible representation of modalities in rings.

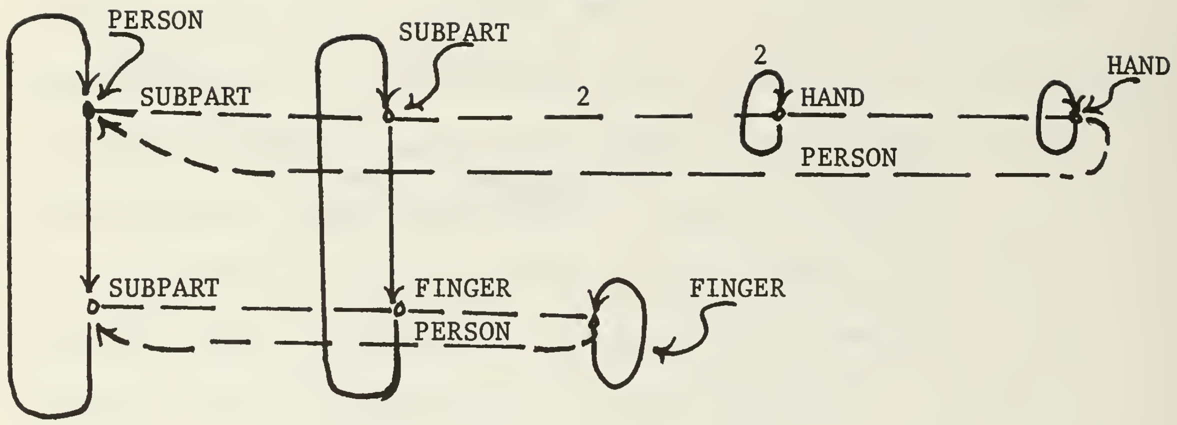

Figure 9. Possible representation of modalities in rings.

SUMMARY

The long range objectives of question answering research are difficult to spell out with precision, because later developments will naturally have considerable influence on the directions taken by the technology. At this time it appears that the lack of expressiveness and flexibility of the data base structures constitutes a primary restriction on the performance of question answering systems. Compared to the data structure problems, the problems of syntax and parsing are well understood.

With the development of better data structures will come more powerful and competent question answering processes. Question answering will merge with problem solving. The long range goal is the development of systems that will serve us as intelligent consultants.

Current Technology

Current question answering systems are able to apply various counting and sorting operations to data that could be simply represented in the form of charts, lists, and tables. In the best systems the linguistic capabilities are more than adequate for this task. However, deductive capabilities are almost totally absent, as are the abilities to solve problems and make generalizations from the data. Question answering processes are designed into the basic structure of the systems, and the systems are unable to discuss these processes with the user or to change them to suit the user's convenience. These systems generally give users the impression that

they are very rigid and unforgiving, use language in strange ways, and are difficult to use. Users get the impression that they must conform to the system, rather than the reverse.

Research Problems

The problem of scale. The systems that provide the most interesting capabilities generally make large demands for computer power, in both time and memory. Consequently it has not been possible to bring the best question answering capabilities available to bear in trial applications of realistic size. It is not clear how expensive of computer power intelligent processes need to be; perhaps they are intrinsically expensive. In any case, it is necessary to investigate ways to apply current question answering capabilities to large data bases. This problem will probably never be solved; there will probably always be applications that are just too large to be accommodated by available systems.

Data Structures. The examination of example data structures earlier in this paper showed that the diversity and complexity of information representable in those structures is limited. This limitation is a critical obstacle to the development of competent question answering systems. The addition of processes as data objects would extend the diversity of data, and also perhaps open the way to putting the question answering processes themselves in to the data base and making them available for discussion and modification by the user. This direction of research would also ease the eventual introduction of true problem solving processes into question answering systems.

Applications. Question answering research could be stimulated and would receive much valuable direction if there were more systems in actual use. In addition, potential users who do not now have question answering systems at their disposal could benefit from increased capabilities such systems could offer them. But the potential user who's operation could be greatly enhanced by a question answering system and the scientist who could provide a suitable system often know nothing of each other's problems and capabilities. An effective way to bridge the gap would be to form teams of people selected from both groups. Such a team could scout for possible applications, evaluate the potential benefit, and provide a pilot system in a short time, if appropriate.

References

1. Craig, James A., et. al., "DEACON: Direct English Access and Control," in Proceedings, Fall Joint Computer Conference, 1966.

2. Dostert, B. H., REL - An Information System for a Dynamic Environment, REL Report Number 3, California Institute of Technology, 1971.

3. Dostert, B. H., and Thompson, F. B., The Syntax of REL English, REL Report Number 1, California Institute of Technology, 1971.

4. Fikes, Richard Earl, "REF-ARF: A System for Solving Problems Stated as Nondeterministic Procedures," J. Art. Intel. 1 (1), 1970.

5. Gibbons, G. D., Beyond REF-ARF: Toward an Intelligent Processor for a Nondeterministic Programming Language, Doctoral dissertation, Carnegie-Mellon University, 1972.

6. Green, Cordell, "Theorem Proving by Resolution as a Basis for Question Answering Systems," Machine Intelligence 4, D. Michie and B. Meltzer, eds. Edinburgh University Press, 1969.

7. Hewitt, Carl, "PLANNER: A Language for Proving Theorems in Robots," Proceedings International Joint Conference of Artificial Intelligence, 1969.

8. Minsky, Marvin, Semantic Information Processing, MIT Press, 1968.

9. Quillian, M. Ross, Semantic Memory, in Minsky, 1968.

10. Raphael, Bertran, "A Computer Program which Understands," in Proceedings, Fall Joint Computer Conference, 1964.

11. Raphael, Bertram, "SIR: Semantic Information Retrieval," in Minsky, 1968.

12. Winograd, Terry, Procedures as a Representation for Data in a Computer Program for Understanding Natural Language, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1971.

13. Hayes, P. J., "A Logic of Actions," in Machine Intelligence VI, Meltzer, B. and Michie, D., eds., American Elsevier Publishing Co., 1971.

DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Dean J. M. Wozencraft Dean of Research, Code 023 Naval Postgraduate School Monterey, California 93940	2
4.	Technical Library, Code 6720 Naval Electronic Laboratory Center San Diego, California 92152	1
5.	Mrs. Dana Small, Code 5000 Naval Electronic Laboratory Center San Diego, California 92152	2
6.	Dr. Douglas W. Gage, Code 3200 Naval Electronic Laboratory Center San Diego, California 92152	1
7.	Dr. Donald O. Christy, Code 3200 Naval Electronic Laboratory Center San Diego, California 92152	1
8.	Dr. R. C. Kolb, Code 3300 Naval Electronic Laboratory Center San Diego, California 92152	1
9.	Mr. H. F. Wong, Code 3200 Naval Electronic Laboratory Center San Diego, California 92152	1
10.	Mr. C. R. Allen, Code 3400 Naval Electronics Laboratory Center San Diego, California 92152	1
11.	Mr. M. A. Lamendola Naval Electronics Laboratory Center San Diego, California 92152	1

		No. Copies
12.	Dr. C. P. Haber, Code 210 Naval Electronics Laboratory Center San Diego, California 92152	1
13.	Dr. E. S. Stewart Naval Electronics Laboratory Center San Diego, California 92152	1
14.	Mr. J. S. Dodds, Code 5000 Naval Electronics Laboratory Center San Diego, California 92152	1
15.	Mr. J. Wolff, Code 20 Navy Personnel and Training Research Laboratory San Diego, California 92152	1
16.	Dr. J. H. Huth, Code 031 Naval Ships Systems Command Washington, D. C. 20360	1
17.	Mr. Marvin Denicoff, Code 437 Department of the Navy Office of Naval Research 800 N. Quincy Street Arlington, Virginia 22217	1
18.	Mr. Gordon D. Goldstein, Code 437 Office of Naval Research 800 N. Quincy Street Arlington, Virginia 22217	1
19.	Dr. Diane Ramsey-Klee R-K Research and System Design 3947 Ridgemont Drive Malibu, California 90265	1
20.	Dr. Bozena Dostert California Institute of Technology 1201 East California Blvd. Pasadena, California 91109	1
21.	Dr. Fred Thompson California Institute of Technology 1201 E. California Blvd. Pasadena, California 91109	1
22.	Dr. David E. Rumelhart Department of Psychology University of California, San Diego P. O. Box 109 La Jolla, California 92037	1

		No. Copies
23.	Dr. E. T. Florance Office of Naval Research 1030 E. Green Street Pasadena, California 91106	1
24.	Dr. George E. Heidorn Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
25.	Dr. Ann Porch South West Regional Laboratory 4661 Lampson Avenue Seal Beach, California 90740	1
26.	Dr. Y. A. Wilks A. I. Project Stanford University Palo Alto, California 94305	1
27.	Dr. Peter Freeman Computer Science Department University of California, Irvine Irvine, California 92664	1
28.	Dr. Richard Fikes A. I. Project Stanford Research Institute Stanford, California 94305	1
29.	Dr. Bertram Raphael A. I. Project Stanford Research Institute Stanford, California 94305	1
30.	Dr. Allen Newell Computer Science Department Carnegie-Mellon University Pittsburgh, Pennsylvania 15213	1
31.	Dr. James Dolby California State University, San Jose San Jose, California 95100	1
32.	Dr. W. M. Woods, Code 53Wo Chairman, Department of Mathematics Naval Postgraduate School Monterey, California 93940	1

		No. Copies
33.	Dr. Gerald L. Barksdale, Jr., Code 72Bv Chairman, Computer Science Group Naval Postgraduate School Monterey, California 93940	1
34.	Dr. Greg Gibbons, Code 53Gp Naval Postgraduate School Monterey, California 93940	6
35.	ENS R. H. Brubaker, Jr., Code 53Bh Naval Postgraduate School Monterey, California 93940	1
36.	LCDR R. M. Hanna, Code 53 Hq Naval Postgraduate School Monterey, California 93940	1
37.	Dr. G. E. Heidorn, Code 55Hd Naval Postgraduate School Monterey, California 93940	1
38.	Dr. G. A. Kildall, Code 53Kd Naval Postgraduate School Monterey, California 93940	1
39.	Dr. U. R. Kodres, Code 53Kr Naval Postgraduate School Monterey, California 93940	1
40.	Dr. L. D. Kovach, Code 53Kv Naval Postgraduate School Monterey, California 93940	1
41.	LT L. G. Litzler, Code 53Lt Naval Postgraduate School Monterey, California 93940	1
42.	LTJG L. R. Moore, III, Code 53Ml Naval Postgraduate School Monterey, California 93940	1
43.	Dr. V. M. Powers, Code 52Pw Naval Postgraduate School Monterey, California	1
44.	Dr. A. M. Shorb, Code 53Sp Naval Postgraduate School Monterey, California 93940	1
45.	Dr. G. H. Syms, Code 53Zz Naval Postgraduate School Monterey, California 93940	1
46.	LTJG T. J. Logan, Code 53Lg Naval Postgraduate School Monterey, California 93940	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY *(Corporate author)* Navy Electronics Laboratory Center	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	
	2b. GROUP	
REPORT TITLE Data Structures for Question Answering Systems		
DESCRIPTIVE NOTES *(Type of report and, inclusive dates)* Technical Report		
AUTHOR(S) *(First name, middle initial, last name)* Gregory D. Gibbons		
REPORT DATE 27 November 1972	7a. TOTAL NO. OF PAGES 55	7b. NO. OF REFS 13
8a. CONTRACT OR GRANT NO. WR2-9100	9a. ORIGINATOR'S REPORT NUMBER(S) NPS-53GP72111A	
b. PROJECT NO.		
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	
d.		
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.		
11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Navy Electronics Laboratory San Diago, California	

13. ABSTRACT

Data bases for question answering systems are models of reality. Such models can be considered to be sets of assertions. Sophisticated models must cope with consistency problems due to the Frame Problem. Processes that derive answers to questions should be included in the model, and question answering systems should be able to discuss such processes. Uniform processes are too rigid to support intelligent question answering systems. Further understanding of nonuniform processes may derive from development of nondeterministic programming languages. The SIR model, the CONVERSE model, and the DEACON model are compared. Directions for research are proposed.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
QUESTION ANSWERING						
DATA BASE						
DATA STRUCTURE						
THE FRAME PROBLEM						
NONDETERMINISTIC PROGRAMMING LANGUAGES						